Peer-Review 2: UML e Protocollo di Comunicazione

Gruppo **GC6** - Rigamonti Alberto, Rogora Matteo, Sarbu Razvan Irinel

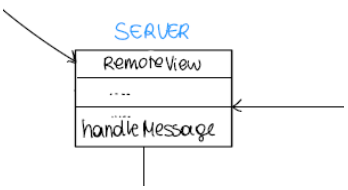
Valutazione del diagramma UML delle classi e del Protocollo di comunicazione del gruppo GC16.

Lati positivi

Nel complesso abbiamo riscontrato molti lati positivi delle implementazioni adottate, soprattutto nei pacchetti contenenti le classi relative al protocollo di comunicazione e quelle relative alla gestione del client.
In particolare, il primo lato positivo che abbiamo notato riguarda la gestione dei messaggi e degli update, che non tentano di inviare il modello nel suo complesso. Sono stati creati diversi messaggi relativi alle varie sezioni del modello che vengono modificate, i quali vengono inviati singolarmente, così da non trasmettere le informazioni in eccesso degli elementi che non variano.
Abbiamo considerato buona la gestione delle lobby multiple attraverso l'utilizzo di UUID.
Un altro elemento positivo è la gestione del "MockModel" sul client, che salva prevalentemente solo le informazioni strettamente necessarie.
Abbiamo infine apprezzato la suddivisione dei messaggi nei quattro tipi ClientMessage, ServerMessage, PlayerEvent e ModelUpdate, che oltre ai vantaggi dal punto di vista funzionale permette una maggior chiarezza nell'interpretazione dei problemi, anche in modalità di Debug.

Lati negativi

Abbiamo trovato alcuni lati negativi di strutturazione dei nuovi componenti aggiunti rispetto alla peer review precedente.



In particolare abbiamo notato una gestione poco chiara degli eventi da parte del controller, ovvero risulta incompleta la descrizione del protocollo di comunicazione riguardante l'Handler dei messaggi e degli eventi ricevuti dal server e dal client.

Un lato relativamente negativo è la gestione del server controller in un'unica classe, con numerosi metodi che rendono poco ordinata la struttura generale e potrebbero creare difficoltà nell'individuazione di problemi durante il testing.

Per la parte restante non abbiamo considerato errate scelte di implementazione differenti, come l'utilizzo dell'interfaccia "IProcessablePacket", in quanto rispecchiano semplicemente una visione alternativa di realizzazione del componente atto a gestire la comunicazione tra messaggi.

Confronto tra le architetture

La gestione delle lobby multiple attraverso il lobby controller è un elemento in comune tra le architetture, in quanto anche il nostro gruppo sta implementando questa funzionalità. La gestione della classe stessa però è diversa: nel nostro caso una lobby è identificata da un codice di 'n' caratteri alfanumerici. Quando un client si connette può scegliere se creare o unirsi a una lobby. Quando una lobby è al completo allora la partita ha inizio e nessuno può connettersi.
Un'altro aspetto che differisce dalla nostra architettura è la gestione degli eventi. Nel nostro caso esiste una classe manager statica che gestisce tutte le funzioni di callback. Quando una lobby viene creata, questa registra le sue funzioni all'interno del manager. Quando arriva un evento da un client, esso viene distribuito in base ad un filtro in modo tale da essere inoltrato alla lobby giusta. Le funzioni di callback di eventi sono identificate dall'annotation @EventHandler.

```
public class EventManager {
   public static synchronized void register(EventListener listenerInstance)
{
       // Register function implementation
   }

   public static synchronized void notify(final Event event) {
       // Notify function implementation
   }

   // Rest of implementation
}

public class Message extends Event {
   private final String message;

   public Message(final String message) {
       this.message = message;
   }

   public String content() {
       return message;
   }
}
```

```java
public class ExampleListener implements EventListener {
    public ExampleListener() {
        EventManager.register(this);
    }

    @EventHandler
    public void onMessage(Message message) {
        System.out.printf("Received new message: %s\n", message.content());
    }
}

public class Main {
    public static void main(String... args) {
        ExampleListener listener = new ExampleListener();
        EventManager.notify(new Message("test"));
    }
}
```

Eseguendo il codice si avrà il seguente risultato

```
Received new message: test
```

In conclusione, abbiamo rilevato pochi lati negativi nell'UML fornito dal gruppo GC16 ma molte differenze di implementazione delle restanti parti del MVC.